Remarks

The Applicant has received and reviewed the Official Action dated 1 March 2007 (the "Action"), and submits this paper as a fully-responsive reply thereto.

The Applicant respectfully requests reconsideration and favorable action on the subject application. Claims 1-2, 4-6, 8, 11-16 and 20 are pending after entry of the revisions indicated above.

Claim Rejections under 35 U.S.C. § 103

As stated in Paragraph 2 of the Action, claims 1, 2, 4-9, 11-16 and 20 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over allegedly admitted prior art (hereinafter, "APA") in view of U.S. Patent No. 6,678,741 to Northcutt et al. (hereinafter, "Northcutt"). Applicant respectfully traverses the rejections, with the comments below being organized for convenience under appropriate headings.

Request for Reconsideration of APA

On page 3, the Action characterized certain subject matter described on pages 3 and 4 of the Applicant's specification as "admitted prior art" (APA). However, the Applicant respectfully traverses this characterization, for at least the reasons set forth in the previous response. In the interests of conciseness, the Applicant refers to these previous remarks, rather than reproducing them here. In light of these previous remarks, the Applicant requests reconsideration of the characterization of pages 3 and 4 of the Applicant's specification as APA.

The Applicant proceeds with the rest of these remarks without conceding that the APA is prior art against the Applicant's claims. More particularly, the Applicant provides the remarks below while maintaining the above request for reconsideration regarding the APA.

Revisions to Certain Claims

Turning now to **independent claim 1**, the Applicant has amended this claim to clarify certain features related to the recited method. The Applicant submits that the revisions to claim 1 are fully supported at least by Figures 2 and 4 of the Applicant's drawings, and related description in the specification.

Turning to the rejection of claim 1, the Applicant submits that the cited portion of the Applicant's specification indicated by the Office Action as APA neither teaches nor suggests at least the revisions to claim 1. For example, the manual checks described on Page 4 of the specification neither teaches nor suggests a computer-readable storage medium storing instructions for performing the recited automated distribution method. Additionally, the description on Pages 2-3 of the Applicant's specification neither teaches nor suggests the hierarchy or organization of the HDT, OMU, OIU, and ONU recited in the beginning of claim 1. The Applicant also submits that Pages 2-3 of the Applicant's specification does not teach or suggest performing the recited processes of "identifying", "determining", and "updating" the software installed on the OMU, the OIU, and the further OIU.

In addition to the forgoing, the Applicant agrees with the shortcomings of the alleged APA that are stated on Page 3 of the Action. Thus, the Action cited

Northcutt for this teaching. Northcutt pertains generally to a method and apparatus for synchronizing firmware. The Action cited Northcutt for its teaching relating to checking the firmware of a second unit, and synchronizing this firmware with the firmware of a first unit. However, without conceding that Northcutt provides the teaching for which it is cited in the Action, the Applicant submits that Northcutt neither teaches nor suggests at least the revisions to claim 1, and therefore fails to cure the deficiencies of the alleged APA.

Based at least on the foregoing revisions and comments, the Applicant submits that the alleged APA and Northcutt, whether considered alone or in combination, support a § 103 rejection of claim 1. The Applicant thus requests reconsideration and withdrawal of the stated § 103 rejection of claim 1.

Turning to **dependent claims 2, and 4-6**, the Applicant has amended these claims for consistency with claim 1, as well as to recite additional features shown in, for example, Figure 4 of the Applicant's drawings. Thus, the above comments directed to claim 1 apply equally to claims 2, and 4-6, in addition to the features recited in these dependent claims.

Dependent claim 7 is cancelled herein only to expedite prosecution of this matter, and without waiver, prejudice, or disclaimer of the subject matter recited therein.

Turning now to **independent claim 8**, the Applicant has revised it similarly to claim 1, which was discussed above. Therefore, the comments directed above to claim 1 apply equally to claim 8. Based at least on the foregoing revisions and comments, the Applicant submits that the alleged APA and Northcutt, whether considered alone or in combination, support a § 103 rejection of claim 8. The

Applicant thus requests reconsideration and withdrawal of the stated § 103 rejection of claim 8.

Dependent claim 9 is cancelled herein only to expedite prosecution of this matter, and without waiver, prejudice, or disclaimer of the subject matter recited therein.

Turning to **dependent claims 11-14**, the Applicant has amended these claims for consistency with claim 8. Thus, the above comments directed to claim 8 apply equally to claims 11-14, in addition to the features recited in these dependent claims.

Dependent claims 15-16 are carried forward without revisions. The comments directed to claim 8 apply equally to claims 15-16.

Turning now to **independent claim 20**, the Applicant has revised it similarly to claim 1, which was discussed above. Therefore, the comments directed above to claim 1 apply equally to claim 20. Based at least on the foregoing revisions and comments, the Applicant submits that the alleged APA and Northcutt, whether considered alone or in combination, support a § 103 rejection of claim 20. The Applicant thus requests reconsideration and withdrawal of the stated § 103 rejection of claim 20.

Requirement for Information

The Action also included a Requirement for Information under 37 CFR § 1.105 (the "Requirement"). In response to the Requirement, the Applicant's representative provided a summary of the Requirement and the text of Rule 1.105 to the inventors. A response to this request as received from inventor Christopher

Drew is included with this paper. More specifically, Mr. Drew provided additional information relating to certain sections of Rule 1.105, as indicated in the attached e-mail hardcopy marked Exhibit A. Mr. Drew's response to sub-section (a)(1)(iii) refers to an email description of how scripts may operate to perform the features described in this application. This email description is marked Exhibit B for ease of reference.

The Applicant submits Exhibit A and Exhibit B for the consideration of the Office in response to the Requirement. If the Office requires any additional information, the Office is requested to contact the undersigned.

<u>Conclusion</u>

Applicant respectfully requests reconsideration and withdrawal of the rejections of claims 1, 2, 4-9, 11-16 and 20, and favorable action on the subject application. If any issue remains unresolved that would prevent allowance of this case, <u>the Examiner is requested to contact the undersigned attorney to resolve the issue</u>.

Respectfully Submitted,

Date: 31 JUL 07

By: [illegible]
Rocco L. Adornato
Lee & Hayes, pllc
Reg. No. 40,480
(509) 324-9256 ext. 257

EXHIBIT A

Rocky Adornato

From: Drew, Christopher [cd7294@att.com]

Sent: Monday, July 09, 2007 12:08 PM

To: Rocky Adornato

Cc: Bates, Bob - BTG; Edwards, Fred; Musser, Max

Subject: RE: Request for Information from the Examiner in charge of BLS/ATT Matter No. 01118 (LH Matter No. BE1 0092 US)

Mr.. Adornato:

We are in the process of getting the requested information, where applicable.

§ 1.105 Requirements for information.

(a)(1) Responses to highlighted subsections.

(iii) Sending (FedEx) hard copy of email description of how scripts works plus actual scripts. Initial idea started with questions as to why we had to manually touch each of the elements each time a new firmware code was release (due to cost in overtime labor costs to do the manual load and frequency of firmware updates needed at that time).

(iv) None known at time of invention - the invention/idea was to automate manual activity and application was developed to that effect..

(v) None known at time of invention.

(vi) Improvement on the Marconi firmware upload (manual process). Improvement was to automate the process: locate device, determine firmware version on device, compare to current upload, if firmware version on device is earlier than code to be loaded, start upload, once upload was complete, validate new version installed, move on to next device.

(vii) In Use: BellSouth (now AT&T) started using process in March of 2001 after ideation, coding, testing, improving process was complete and we determined that application worked as desired

(viii) Non known. The general invention of auto-upload of software to remote device is network and device non-specific.

Please let me know if you need any additional information.

Christopher Drew
MMCC-Finance
404-986-8629

From: Rocky Adornato [mailto:rocky@LeeHayes.com]
Sent: Monday, July 02, 2007 5:36 PM
To: Musser, Max; Bates, Bob - BTG; Drew, Christopher; Edwards, Fred
Cc: Hartman, Jodi
Subject: Request for Information from the Examiner in charge of BLS/ATT Matter No. 01118 (LH Matter No. BE1 0092 US)

Hello inventors:

This patent application pertains to Systems and Methods for Automated Software Distribution in a Fiber Optic Network, and we are currently prosecuting this application in the United States Patent Office. The Examiner has recently requested that we provide him with certain information relating to statements that were included in the Background section of our application. Here is the basis for this request, as it appears in the patent rules, with yellow highlight for what appears to be the more relevant portions:

§ 1.105 Requirements for information.
(a)(1) In the course of examining or treating a matter in a pending or abandoned application filed under 35 U.S.C. 111 or 371 (including a reissue application), in a patent, or in a reexamination proceeding, the examiner or other Office employee may require the submission, from individuals identified under § 1.56(c), or any assignee, of such information as may be reasonably necessary to properly examine or treat the matter, for example:

(i)*Commercial databases*: The existence of any particularly relevant commercial database known to any of the inventors that could be searched for a particular aspect of the invention.

(ii)*Search*: Whether a search of the prior art was made, and if so, what was searched.

(iii)*Related information*: A copy of any non-patent literature, published application, or patent (U.S. or foreign), by any of the inventors, that relates to the claimed invention.

(iv)*Information used to draft application*: A copy of any non-patent literature, published application, or patent (U.S. or foreign) that was used to draft the application.

(v)*Information used in invention process*: A copy of any non-patent literature, published application, or patent (U.S. or foreign) that was used in the invention process, such as by designing around or providing a solution to accomplish an invention result.

(vi)*Improvements*: Where the claimed invention is an improvement, identification of what is being improved.

(vii)*In Use:* Identification of any use of the claimed invention known to any of the inventors at the time the application was filed notwithstanding the date of the use.

(viii) *Technical information known to applicant.* Technical information known to applicant concerning the related art, the disclosure, the claimed subject matter, other factual information pertinent to patentability, or concerning the accuracy of the examiner's stated interpretation of such items.

...

For ease of reference, I have attached a .pdf of the application as filed in 2001. The Examiner is asking for the above information regarding our background discussion that appears on pages 1-4. In particular, the Examiner asked about any information we have regarding Marconi Communications (as mentioned on page 2, line 14), or the "service providers" (as mentioned on page 3, line 11).

I am available for a conference call if that would help to provide additional information or context for what the Examiner is asking for, and to gauge what type of information we might have on hand to provide.

Thanks,

Rocky

Rocky Adornato
(509)324-9256 x257
rocky@leehayes.com

Lee & Hayes pllc, Intellectual Property Law
421 West Riverside, Suite 500, Spokane, WA 99201 | 509.323-8979 fax | www.leehayes.com

Drew, Christopher

EXHIBIT B

From:	Max Musser [Max Musser /m6,mail6a] on behalf of Max Musser [Max.Musser@bridge.bellsouth.com]
Sent:	Wednesday, January 24, 2001 3:58 PM
To:	Christopher.Drew@bellsouth.com
Subject:	OMU & OIU Upgrade executables

Chris,

This is the procedure that is used to upgrade large numbers on OMUs and OIUs automatically.

1. I run the checkver.perl script (same as the checkver.sh script from Marconi with a few enhancements) with the $ipfilter and $infile lines commented out so that it will run on all OMUs. This produces a report (checkver.out).
2. I move checkver.out to my data directory.
3. I run the checkverdb.perl script to generate a checkver.db file. This file is used to maintain a status of all the OMUs. The status can be one of four possibilities:

```
            All versions OK          All versions match the versions in
the checkver.perl script.
            Incorrect OIU            One or more of the OIU versions is
incorrect.  The OMU version is correct.
            Incorrect OMU            The OMU version is incorrect.  The
OIU versions may or may not be correct.
            Unable to ping           The OMU was unreachable.  The status
is unknown.
```

4. I run the badomus.perl script to generate a list of IPs of all OMUs that have the status "Incorrect OMU". I copy this file to badomus.out.
5. I run the upgrade_omus.perl script. It runs the omudownload script once for every IP in the badomus.out file.
6. Once the script has finished running, I uncomment the $infile line in the checkver.perl script.
7. I copy the badomus.out file to the file assigned to $infile.
8. I run the checkver.perl script.
9. I move checkver.out to my data directory.
10. I run the checkverdb.perl script to update the checkver.db file.
11. I run the badoius.perl script to generate a list of IPs of all OMUs that have the status "Incorrect OIU". I copy this file to badoius.out.
12. I run the upgrade_oius.perl script. It runs the oiudownload script once for every IP in the badoius.out file.
13. Once the script has finished running, I copy the badoius.out file to the file assigned to $infile in the checkver.perl script.
14. I run the checkver.perl script.
15. I move checkver.out to my data directory.
16. I run the checkverdb.perl script to update the checkver.db file.
17. For those OMUs that I was unable to ping earlier I run the noping.perl script. It generates a list of IPs of OMUs with the status "Unable to ping" in the file noping.out.
18. I run the checkver.perl script with $infile set to noping.out.
19. I move checkver.out to my data directory.
20. I run the checkverdb.perl script to update the checkver.db file.
21. I repeat earlier steps as needed to resolve all OMUs that do not have a status of "All versions OK".
22. The checkver.perl script has an $ipfilter variable that allows me to check only OMUs that have IPs that match the filter.
23. The omudownload and oiudownload scripts are copies of the swdownload script provided by Marconi which I have altered by hard coding the .img file.

The differences between this automated process and the manual process are:

1. A database (checkver.db) of the status of the OMUs is maintained.
2. Multiple OMUs are upgraded from a list of IPs using perl scripts.

upgrade_ous.perl upgrade_omus.perl oudownload omudownload checkverdb.perl

3. Scripts are ava

checkver.perl badous.perl noping.perl badomus.perl

generate the IP lists from the database
based on OMU status.

```
#!/usr/local/perl-5.6/bin/perl
#########################################################################
#############
#
# Script:       upgrade_oius.perl
# Author:       MLM
# Last Update:  18Jan01
# Description:  This perl script upgrades OIU software from a the list
of OIUs in
#               badoius.out.
#
#########################################################################
#############

$infile  = "/home/bwczkdj/data/badoius.out";
die "FATAL ERROR:  Unable to open $infile\n" unless -e $infile;
open IN, "$infile";
while (<IN>) {
    chomp;
    $date = scalar localtime;
    print "Upgrading $_ ... $date\n";
    @a = `/home/bwczkdj/scripts/oiudownload $_ << EOT



EOT
`;
}
close IN;
print "upgrade_oius.perl complete.\n";
exit;
```

```
#!/usr/local/perl-5.6/bin/perl
##########################################################################
#############
#
# Script:        upgrade_omus.perl
# Author:        MLM
# Last Update:   18Jan01
# Description:   This perl script upgrades OMU software from a the list
of OMUs in
#                badomus.out.
#
##########################################################################
#############

$infile  = "/home/bwczkdj/data/badomus.out";
die "FATAL ERROR:  Unable to open $infile\n" unless -e $infile;
open IN, "$infile";
while (<IN>) {
    chomp;
    $date = scalar localtime;
    print "Upgrading $_ ... $date\n";
    @a = `/home/bwczkdj/scripts/omudownload $_ << EOT



EOT
`;
}
print "upgrade_omus.perl complete.\n";
exit;
```

```
#!/bin/ksh
#
# A script to perform download on multiple OMU, to be launched within
OpenView
#
# This script assumes the FiberStar MIB has been loaded in OpenView
#
# The logpath is hardcoded to /opt/marconi/javlin/data/log
# The tftp directory is hardcoded to /tftpboot/marconi
#

#
# make sure the IP address or hostname is entered as a command-line
argument
#
if [ $# = "0" ]
then
  echo "Error: missing IP/hostname argument"
  echo "Usage: swdownload IP/hostname"
  echo
  exit
fi

#
# set up the log file
#
logfile=swdownload$$.log
logpath=/opt/marconi/javlin/data/log
if [ ! -d $logpath ]; then
  mkdir $logpath
fi
echo "*** Download log can be found at $logpath/$logfile"
echo
echo "Log file for FiberStar software download" >> $logpath/$logfile
echo "Created:  $(date)" >> $logpath/$logfile
echo >> $logpath/$logfile

#
# prompt for the set community string
#
setstring="public"
echo
echo "Please enter the SNMP set community string: [$setstring] \c"
read newstring
if [ ${#newstring} != 0 ]; then
  setstring=$newstring
fi

# log message
echo "set community string specified is: $setstring" >>
$logpath/$logfile

#
# prompt for a directory
#
targetdir=/tftpboot/marconi
dirisgood=0
while [ $dirisgood == 0 ]; do
  echo
  echo "Please enter the source directory of the software image files:
[$targetdir]"
```

```
  read newdir
  if [ ${#newdir} != 0 ]; then
    targetdir=$newdir
  fi
  if [ ! -e $targetdir ]; then
    echo "Error: Directory entered does not exist, try again."
  elif [ ${#targetdir} == 0 ]; then
    echo "Error: No directory is entered, try again."
  else
    dirisgood=1
    echo
  fi
done

# log message
echo "directory specified is: $targetdir" >> $logpath/$logfile

cd $targetdir

# log message
echo "image file selected is: oiu12.img" >> $logpath/$logfile

#
# perform download on selected OMU
#
for omu in $@
do
  noskip=0
  answer='y'
  echo
  echo "Download file oiu12.img to \"$omu\" ? [y] \c"
  read newanswer
  if [ ${#newanswer} != 0 ]; then
    answer=$newanswer
  fi
  if [ $answer = 'y' ]; then
    echo "\nDownloading file oiu12.img to \"$omu\" " >>
$logpath/$logfile
    echo "Please wait while preparing for download..\c"
# set the filename to do tftp get on, show snmpset error messages if an
error
# occurs
    /opt/OV/bin/snmpset -c $setstring $omu tftpDownloadFilename.0
octetstringascii "oiu12.img" >> $logpath/$logfile 2>> $logpath/$logfile
    if [ $? != 0 ]; then
      echo
      echo "Error: failed to set the tftpDownloadFilename variable"
      echo "       check network or SNMP configuration"
      echo "       download will be skipped on $omu"
# if filename is not set correctly, skip this OMU
      noskip=1
    else
      echo "..\c"
    fi

# kick off the tftp get by setting tftpStatus variable to "start-
tftp(2)"
    if [ $noskip == 0 ]; then
      /opt/OV/bin/snmpset -c $setstring $omu tftpStatus.0 integer 2 >>
$logpath/$logfile 2>> $logpath/$logfile
      if [ $? != 0 ]; then
```

```
          echo
          echo "Error: failed to set the tftpStatus variable"
          echo "       check network or SNMP configuration"
          echo "       download will be skipped on $omu"
# if action status is not set correctly, skip this OMU
          noskip=1
        else
          echo ".."
        fi
      fi
    else
# move to the next OMU
      noskip=1
    fi

#
# show the download progress and check the tftpStatus periodically
#
if [ $noskip == 0 ]; then
  echo
  echo "Depending on network traffic, this process could take a few
minutes..."
  echo "Downloading\c"
  counterA=0
# wait altogether for 30 minutes
  while [[ $counterA -lt 30 ]]; do
    counterB=0
# wait for 60 seconds before checking the tftp status
    while [[ $counterB -lt 15 ]]; do
      sleep 4
      echo ".\c"
      let counterB=counterB+1
    done
    ret=$(/opt/OV/bin/snmpget -c $setstring $omu tftpStatus.0 |cut -d: -
f3 |cut -c2- 2>> $logpath/$logfile)
    if [ -z $ret ]; then
# timeout or other SNMP error occurred, skip further checking
      ret=snmp-error
      break
    fi
    if [ $ret != "tftp-in-progress" ]; then
      break
    fi
    let counterA=counterA+1
  done

  echo
  echo

#
# check the download result, report any error condition
#
  if [ $ret == "no-error" ]; then
    echo "Download to $omu is successful"
    echo "Download to $omu is successful" >> $logpath/$logfile
  elif [ $ret == "tftp-unknown-error" ]; then
    echo "Error: tftp unknown error, download to $omu failed"
    echo "Error: tftp unknown error, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-file-not-found" ]; then
    echo "Error: tftp file not found, download to $omu failed"
    echo "Error: tftp file not found, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-access-violation" ]; then
    echo "Error: tftp access violation, download to $omu failed"
    echo "Error: tftp access violation, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-disk-full" ]; then
    echo "Error: tftp disk full, download to $omu failed"
    echo "Error: tftp disk full, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-illegal-operation" ]; then
    echo "Error: tftp illegal operation, download to $omu failed"
    echo "Error: tftp illegal operation, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-unknown-xfer-id" ]; then
    echo "Error: tftp unknown xfer id, download to $omu failed"
    echo "Error: tftp unknown xfer id, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-file-exists" ]; then
    echo "Error: tftp file exists, download to $omu failed"
    echo "Error: tftp file exists, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-no-such-user" ]; then
    echo "Error: tftp no such user, download to $omu failed"
    echo "Error: tftp no such user, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "bad-file" ]; then
    echo "Error: bad file, download to $omu failed"
    echo "Error: bad file, download to $omu failed" >> $logpath/$logfile
  elif [ $ret == "flash-program-failure" ]; then
    echo "Error: flash program failure, download to $omu failed"
    echo "Error: flash program failure, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "oiu-download-failure" ]; then
    echo "Error: OIU download failure, this probably happened on 1 of
the possible 8 OIUs"
    echo "       check the onuDescr variable in the MIB browser to
verify the configuration"
    echo "       and download progress"
    echo "Error: OIU download failure, this probably happened on 1 of
the possible 8 OIUs" >> $logpath/$logfile
    echo "       check the onuDescr variable in the MIB browser to
verify the configuration" >> $logpath/$logfile
    echo "       and download progress" >> $logpath/$logfile
  elif [ $ret == "snmp-error" ]; then
    echo "Error: timeout or other SNMP error on $omu,"
    echo "       check download status or retry when error condition is
cleared"
    echo "Error: timeout or other SNMP error on $omu" >>
$logpath/$logfile
    echo "       check download status or retry when error condition is
cleared" >> $logpath/$logfile
  elif [ $ret == "tftp-in-progress" ]; then
    echo "Download is not completed yet after 30 mins., please use the
MIB browser"
    echo "to verify the download result on the \"tftpStatus\" variable"
    echo "The status checking phase is terminated"
    echo "Download is not completed yet after 30 mins., please use the
MIB browser" >> $logpath/$logfile
    echo "to verify the download result on the \"tftpStatus\" variable"
>> $logpath/$logfile
```

```
    echo "The status checking phase is terminated" >> $logpath/$logfile
  fi
#end of noskip
fi

done
# end of for omu in $@

echo
echo "Download process is completed for the selected OMU,"
echo "check the log file $logpath/$logfile for detailed information."
echo "Press <return> to exit..."
read
```

```
#!/bin/ksh
#
# A script to perform download on multiple OMU, to be launched within
OpenView
#
# This script assumes the FiberStar MIB has been loaded in OpenView
#
# The logpath is hardcoded to /opt/marconi/javlin/data/log
# The tftp directory is hardcoded to /tftpboot/marconi
#


#
# make sure the IP address or hostname is entered as a command-line
argument
#
if [ $# = "0" ]
then
  echo "Error: missing IP/hostname argument"
  echo "Usage: swdownload IP/hostname"
  echo
  exit
fi

#
# set up the log file
#
logfile=swdownload$$.log
logpath=/opt/marconi/javlin/data/log
if [ ! -d $logpath ]; then
  mkdir $logpath
fi
echo "*** Download log can be found at $logpath/$logfile"
echo
echo "Log file for FiberStar software download" >> $logpath/$logfile
echo "Created:  $(date)" >> $logpath/$logfile
echo >> $logpath/$logfile

#
# prompt for the set community string
#
setstring="public"
echo
echo "Please enter the SNMP set community string: [$setstring] \c"
read newstring
if [ ${#newstring} != 0 ]; then
  setstring=$newstring
fi

# log message
echo "set community string specified is: $setstring" >>
$logpath/$logfile

#
# prompt for a directory
#
targetdir=/tftpboot/marconi
dirisgood=0
while [ $dirisgood == 0 ]; do
  echo
  echo "Please enter the source directory of the software image files:
 [$targetdir]"
```

```
  read newdir
  if [ ${#newdir} != 0 ]; then
    targetdir=$newdir
  fi
  if [ ! -e $targetdir ]; then
    echo "Error: Directory entered does not exist, try again."
  elif [ ${#targetdir} == 0 ]; then
    echo "Error: No directory is entered, try again."
  else
    dirisgood=1
    echo
  fi
done

# log message
echo "directory specified is: $targetdir" >> $logpath/$logfile

cd $targetdir

# log message
echo "image file selected is: omu12.img" >> $logpath/$logfile

#
# perform download on selected OMU
#
for omu in $@
do
  noskip=0
  answer='y'
  echo
  echo "Download file omu12.img to \"$omu\" ? [y] \c"
  read newanswer
  if [ ${#newanswer} != 0 ]; then
    answer=$newanswer
  fi
  if [ $answer = 'y' ]; then
    echo "\nDownloading file omu12.img to \"$omu\" " >>
$logpath/$logfile
    echo "Please wait while preparing for download..\c"
# set the filename to do tftp get on, show snmpset error messages if an
error
# occurs
    /opt/OV/bin/snmpset -c $setstring $omu tftpDownloadFilename.0
octetstringascii "omu12.img" >> $logpath/$logfile 2>> $logpath/$logfile
    if [ $? != 0 ]; then
      echo
      echo "Error: failed to set the tftpDownloadFilename variable"
      echo "       check network or SNMP configuration"
      echo "       download will be skipped on $omu"
# if filename is not set correctly, skip this OMU
      noskip=1
    else
      echo "..\c"
    fi

# kick off the tftp get by setting tftpStatus variable to "start-
tftp(2)"
    if [ $noskip == 0 ]; then
      /opt/OV/bin/snmpset -c $setstring $omu tftpStatus.0 integer 2 >>
$logpath/$logfile 2>> $logpath/$logfile
      if [ $? != 0 ]; then
```

```
          echo
          echo "Error: failed to set the tftpStatus variable"
          echo "       check network or SNMP configuration"
          echo "       download will be skipped on $omu"
# if action status is not set correctly, skip this OMU
          noskip=1
        else
          echo ".."
        fi
      fi
    else
# move to the next OMU
      noskip=1
    fi


#
# show the download progress and check the tftpStatus periodically
#
if [ $noskip == 0 ]; then
  echo
  echo "Depending on network traffic, this process could take a few
minutes..."
  echo "Downloading\c"
  counterA=0
# wait altogether for 30 minutes
  while [[ $counterA -lt 30 ]]; do
    counterB=0
# wait for 60 seconds before checking the tftp status
    while [[ $counterB -lt 15 ]]; do
      sleep 4
      echo ".\c"
      let counterB=counterB+1
    done
    ret=$(/opt/OV/bin/snmpget -c $setstring $omu tftpStatus.0 |cut -d: -
f3 |cut -c2- 2>> $logpath/$logfile)
    if [ -z $ret ]; then
# timeout or other SNMP error occurred, skip further checking
      ret=snmp-error
      break
    fi
    if [ $ret != "tftp-in-progress" ]; then
      break
    fi
    let counterA=counterA+1
  done

  echo
  echo


#
# check the download result, report any error condition
#
  if [ $ret == "no-error" ]; then
    echo "Download to $omu is successful"
    echo "Download to $omu is successful" >> $logpath/$logfile
  elif [ $ret == "tftp-unknown-error" ]; then
    echo "Error: tftp unknown error, download to $omu failed"
    echo "Error: tftp unknown error, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-file-not-found" ]; then
    echo "Error: tftp file not found, download to $omu failed"
    echo "Error: tftp file not found, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-access-violation" ]; then
    echo "Error: tftp access violation, download to $omu failed"
    echo "Error: tftp access violation, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-disk-full" ]; then
    echo "Error: tftp disk full, download to $omu failed"
    echo "Error: tftp disk full, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-illegal-operation" ]; then
    echo "Error: tftp illegal operation, download to $omu failed"
    echo "Error: tftp illegal operation, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-unknown-xfer-id" ]; then
    echo "Error: tftp unknown xfer id, download to $omu failed"
    echo "Error: tftp unknown xfer id, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-file-exists" ]; then
    echo "Error: tftp file exists, download to $omu failed"
    echo "Error: tftp file exists, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "tftp-no-such-user" ]; then
    echo "Error: tftp no such user, download to $omu failed"
    echo "Error: tftp no such user, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "bad-file" ]; then
    echo "Error: bad file, download to $omu failed"
    echo "Error: bad file, download to $omu failed" >> $logpath/$logfile
  elif [ $ret == "flash-program-failure" ]; then
    echo "Error: flash program failure, download to $omu failed"
    echo "Error: flash program failure, download to $omu failed" >>
$logpath/$logfile
  elif [ $ret == "oiu-download-failure" ]; then
    echo "Error: OIU download failure, this probably happened on 1 of
the possible 8 OIUs"
    echo "       check the onuDescr variable in the MIB browser to
verify the configuration"
    echo "       and download progress"
    echo "Error: OIU download failure, this probably happened on 1 of
the possible 8 OIUs" >> $logpath/$logfile
    echo "       check the onuDescr variable in the MIB browser to
verify the configuration" >> $logpath/$logfile
    echo "       and download progress" >> $logpath/$logfile
  elif [ $ret == "snmp-error" ]; then
    echo "Error: timeout or other SNMP error on $omu,"
    echo "       check download status or retry when error condition is
cleared"
    echo "Error: timeout or other SNMP error on $omu" >>
$logpath/$logfile
    echo "       check download status or retry when error condition is
cleared" >> $logpath/$logfile
  elif [ $ret == "tftp-in-progress" ]; then
    echo "Download is not completed yet after 30 mins., please use the
MIB browser"
    echo "to verify the download result on the \"tftpStatus\" variable"
    echo "The status checking phase is terminated"
    echo "Download is not completed yet after 30 mins., please use the
MIB browser" >> $logpath/$logfile
    echo "to verify the download result on the \"tftpStatus\" variable"
>> $logpath/$logfile
```

```
    echo "The status checking phase is terminated" >> $logpath/$logfile
  fi
#end of noskip
fi

done
# end of for omu in $@

echo
echo "Download process is completed for the selected OMU,"
echo "check the log file $logpath/$logfile for detailed information."
echo "Press <return> to exit..."
read
```

```
#!/usr/local/perl-5.6/bin/perl
##########################################################################
#############
#
# Script:       checkver.perl
# Author:       MLM
# Last Update:  02Jan01
# Description:  This perl script checks the version of all FiberStar
OMU-18s and
#               their OIU-4Xs.
#
##########################################################################
#############

#-------------------------------------------------------------------------
-------------
# user-modifiable variables
#-------------------------------------------------------------------------
-------------
@correctomuvers = qw(1.2);
@correctoiuvers = qw(1.2);

$ping_timeout  = 1;

$outfile  = "/tmp/checkver.out";
$infile   = "/home/bwczkdj/data/noping.out";
#$ipfilter = "24\.14\.";      # use if you want only a subset of OMUs

$ovdir   = "/opt/OV/bin";
$ovbin   = "/opt/OV/bin";
$dhcpdir = "/var/dhcp";

#-------------------------------------------------------------------------
-------------
# non user-modifiable variables
#-------------------------------------------------------------------------
-------------
foreach (@correctomuvers) {
    push (@omuverstrs, "FIBERSTAR OMU18 Version $_");
}
foreach (@correctoiuvers) {
    push (@oiu45verstrs, "Ports1-4: FIBERSTAR OIU45(4 Port) Version
$_");
    push (@oiu48verstrs, "Ports1-4: FIBERSTAR OIU48(8 Port) Version $_,
"
                       . "Ports5-8: FIBERSTAR OIU48(8 Port) Version
$_");
}

$nopingfile  = "/tmp/noping.$$";
$badverfile  = "/tmp/badver.$$";
$okomufile   = "/tmp/okomus.$$";

# display banner to STDOUT
#------------------------
print "\n";
print
"************************************************************************
*\n";
print "           FiberStar OMU-18/OIU-48/OIU-45 Version Checker\n";
print
```

```
"************************************************************************
*\n";
print "\n";

# write banner to $outfile
#-------------------------
open  LOG, ">$outfile";
print LOG
"************************************************************************
*\n";
print LOG "              FiberStar OMU-18/OIU-48/OIU-45 Version
Checker\n";
print LOG
"************************************************************************
*\n";
print LOG "\n";
print LOG scalar localtime;
print LOG "\n";

# set up HPOV environment
#-------------------------
$get = "$ovbin/snmpget";
die "FATAL ERROR:  \$ovbin varible is not correct\n" unless -e $get;

die "FATAL ERROR:  \$infile ($infile) does not exist\n" unless (-e
$infile || $infile eq "");

# get list of OMU-18s
#--------------------
print "Creating list of OMUs ... ";

# put list of OMU-18s in $omuiplist file
#
# this is done by searching for the Marconi MAC address "00C09B"
# in DHCP configuration files;  if any OMUs have a MAC address that
# starts with a different value, they won't be found
#-----------------------------------------------------------------
@filelist = (<$dhcpdir/*>);
foreach $i (@filelist) {
    open DHCPFILE, $i;
    while (<DHCPFILE>) {
        if (/^0100C09B/) {
            @b = split;
            $iphash{"$b[2]"} = $b[5] if $b[2] =~ /^$ipfilter/;
        }
    }
    close DHCPFILE;
}

if (-e $infile) {
    open IPLIST, "$infile";
    @iplist = <IPLIST>;
    close IPLIST;
    foreach (@iplist) {
        chomp;
    }
    $num_omus = scalar @iplist;
    print "Found $num_omus OMU-18s in $infile\n\n";
    print LOG "Found $num_omus OMU-18s in $infile\n\n";
} else {
    @iplist = sort { $a <=> $b } (keys %iphash);
```

```
        $num_omus = scalar @iplist;
        print "Found $num_omus unique OMU-18s in $dhcpdir files\n\n";
        print LOG "Found $num_omus unique OMU-18s in $dhcpdir files\n\n";
    }

    open NOPING, ">$nopingfile";
    open BADVER, ">$badverfile";
    open OKOMU,  ">$okomufile";

    # check each OMU
    #---------------
    foreach (@iplist) {

        # get mnemonic name for OMU IP address
        #-------------------------------------
        $omu_name = $iphash{$_};
        next if $omu_name eq "";
        print "$_ ($omu_name) ... ";

        # ping OMU
        #---------
        $ping = `/usr/sbin/ping $_ $ping_timeout`;
        if ($ping =~ /^no answer/) {
            print "Unable to ping\n";
            print NOPING "Unable to ping OMU $_ ($omu_name)\n";
        } else {
            # OMU is up;  check its version strings
            #--------------------------------------
            $all_ok = 0;

            # get segDescr for OMU
            #---------------------
            $verx = `$get $_ reltec.fiberstar.segment.segDescr.0`;
            chomp $verx;
            $ver = substr($verx, index($verx, "(ascii)") + 10);
            foreach (@omuverstrs) {
                $all_ok = 1 if $ver eq $_;
            }
            print BADVER "Incorrect OMU version for OMU $_ ($omu_name):
 $ver\n" if $all_ok == 0;

            # check each OIU-4x
            #------------------
            for ($i = 1; $i <= 8; $i++) {
                $verx = `$get $_
 reltec.fiberstar.segment.onu.onuTable.onuEntry.onuDescr.$i`;
                chomp $verx;
                $ver = substr($verx, index($verx, "(ascii)") + 10);
                $modx = `$get $_
 reltec.fiberstar.segment.onu.onuTable.onuEntry.onuOIUModel.$i`;
                chomp $modx;
                $mod = substr($modx, index($modx, "INTEGER") + 9);

                $match = 0;
                if ($mod eq "oiu48") {
                    foreach (@oiu48verstrs) {
                        $match = 1 if $ver eq $_;
                    }
                }
                if ($mod eq "oiu45") {
                    foreach (@oiu45verstrs) {
```

```
                    $match = 1 if $ver eq $_;
                }
            }
            $match = 1 if ($mod ne "oiu48") && ($mod ne "oiu45");
            if ($match == 0) {
                $all_ok = 0;
                print BADVER "Incorrect OIU-4x version for OMU $_
($omu_name), "
                             . "ONU #$i: $ver\n";
            }
        }
        # see if OMU passed all checks
        #---------------------------
        if ($all_ok == 1) {
            print "All versions OK\n";
            print OKOMU "$_ ($omu_name):  All versions OK\n";
        } else {
            print "******** Unexpected versions found *******\n";
        }
    }
}

close NOPING;
close BADVER;
close OKOMU;

# create $outfile from $okomufile, $nopingfile, and $badverfile
#---------------------------------------------------------------
print LOG "----------------\n";
print LOG "Expected versions\n";
print LOG "----------------\n";
$size = scalar @correctomuvers;

print LOG "OMU   :  ";
if ($size == 1) {
    print LOG "$correctomuvers[0]\n";
} else {
    for ($i = 0; $i < $size; $i++) {
        print LOG $correctomuvers[$i];
        print LOG ", " if $i <= ($size - 2);
        print LOG "or " if $i == ($size - 2);
    }
    print LOG "\n";
}
$size = scalar @correctoiuvers;
print LOG "OIU-45:  ";
if ($size == 1) {
    print LOG "$correctoiuvers[0]\n";
} else {
    for ($i = 0; $i < $size; $i++) {
        print LOG $correctoiuvers[$i];
        print LOG ", " if $i <= ($size - 2);
        print LOG "or " if $i == ($size - 2);
    }
    print LOG "\n";
}
$size = scalar @correctoiuvers;
print LOG "OIU-48:  ";
if ($size == 1) {
    print LOG "$correctoiuvers[0]\n";
} else {
        for ($i = 0; $i < $size; $i++) {
            print LOG $correctoiuvers[$i];
            print LOG ", " if $i <= ($size - 2);
            print LOG "or " if $i == ($size - 2);
        }
        print LOG "\n\n";
    }
    print LOG "----------------------------\n";
    print LOG "OMUs with unexpected versions\n";
    print LOG "----------------------------\n";

    if (-e $badverfile) {
        open BADVER, "$badverfile";
        while (<BADVER>) {
            print LOG $_;
        }
        close BADVER;
    }

    print LOG "\n";
    print LOG "-------------------------------------------------------------
--\n";
    print LOG "OMUs which did not respond to pings (ping timeout =
$ping_timeout seconds)\n";
    print LOG "-------------------------------------------------------------
--\n";

    if (-e $nopingfile) {
        open NOPING, "$nopingfile";
        while (<NOPING>) {
            print LOG $_;
        }
        close NOPING;
    }

    print LOG "\n";
    print LOG "-----------------------\n";
    print LOG "OMUs which appear normal\n";
    print LOG "-----------------------\n";

    if (-e $okomufile) {
        open OKOMU, "$okomufile";
        while (<OKOMU>) {
            print LOG $_;
        }
        close OKOMU;
    }

    print LOG "\n";
    $date = scalar localtime;
    print LOG
"*********************************************************************
*\n";
    print LOG " $date:  Version checking complete.\n";
    print LOG
"*********************************************************************
*\n";
    close LOG;

    # clean up
    #---------
```

```
unlink $nopingfile;
unlink $badverfile;
unlink $okomufile;

print "\n";
print
"***********************************************************************
*\n";
print " Version checking complete.  See $outfile for more details.\n";
print
"***********************************************************************
*\n\n";

exit;
```

```
#!/usr/local/perl-5.6/bin/perl
################################################################################
#############
#
# Script:         badoius.perl
# Author:         MLM
# Last Update:    18Jan01
# Description:    This perl script prints a list of bad OIUs from the
checkver.out
#                 file.
#
################################################################################
#############

$infile   = "/home/bwczkdj/data/checkver.db";
$outfile  = "/home/bwczkdj/data/noping.out";
die "FATAL ERROR:  Unable to open $infile\n" unless -e $infile;
open IN, "$infile";
open OUT, ">$outfile";
while (<IN>) {
    if (/Incorrect OIU/) {
        @temp = split /./;
        $ip = $temp[0];
        print OUT "$ip\n";
    }
}
print "badoius.perl complete.\n";
exit;
```

```
#!/usr/local/perl-5.6/bin/perl
##########################################################################
#############
#
# Script:        noping.perl
# Author:        MLM
# Last Update:   02Jan01
# Description:   This perl script reads the checkver.db file
#                and generates a list of IPs that could not be pinged.
#
##########################################################################
#############

$infile  = "/home/bwczkdj/data/checkver.db";
$outfile = "/home/bwczkdj/data/noping.out";

open (DB, "$infile") or die "FATAL ERROR:  Unable to open $infile";
open (OUT, ">$outfile") or die "FATAL ERROR:  Unable to open $outfile";


while (<DB>) {
    if (/Unable/) {
        @b = split /,/;
        print OUT "$b[0]\n";
    }
}
close DB;
close OUT;

print "noping.perl complete.\n";
```

```
#!/usr/local/perl-5.6/bin/perl
###########################################################################
##############
#
# Script:        badomus.perl
# Author:        MLM
# Last Update:   18Jan01
# Description:   This perl script prints a list of bad OMUs from the
checkver.out
#                file.
#
###########################################################################
##############

$infile   = "/home/bwczkdj/data/checkver.db";
$outfile  = "/home/bwczkdj/data/noping.out";
die "FATAL ERROR:  Unable to open $infile\n" unless -e $infile;
open IN, "$infile";
open OUT, ">$outfile";
while (<IN>) {
    if (/Incorrect OMU/) {
        @temp = split /,/;
        $ip = $temp[0];
        print OUT "$ip\n";
    }
}
print "badomus.perl complete.\n";
exit;
```

```
#!/usr/local/perl-5.6/bin/perl
########################################################################
#############
#
# Script:       checkverdb.perl
# Author:       MLM
# Last Update:  19Jan01
# Description:  This perl script updates the checkver.db database for
the FiberStar
#               OMUs and OIUs.
#
########################################################################
##############

$infile     = "/home/bwczkdj/data/checkver.out";
$dbfilename = "/home/bwczkdj/data/checkver.db";

die "FATAL ERROR:  Unable to open $infile\n" unless -e $infile;

if (-e $dbfilename) {
    open DB, "$dbfilename";
    @a = <DB>;
    close DB;
    shift @a; shift @a;
    foreach (@a) {
        chomp;
        @b = split /,/;
        $omu_name{$b[0]}    = $b[1];
        $status{$b[0]}      = $b[2];
        $bad_omu{$b[0]}     = 0;
    }
}

open IN, "$infile";
open DB, ">$dbfilename";

print DB "IP,OMU NAME,STATUS\n";
print DB "--,--------,------\n";

while (<IN>) {
    chomp;
    @temp = split;
    if (/^Unable/) {
        $ip = $temp[4];
        $omu_name{$ip} = $temp[5];
        $status{$ip} = "Unable to ping" if $status{$ip} eq "";
    } elsif (/^Incorrect OIU/) {
        $ip = $temp[5];
        chop $temp[6];
        $omu_name{$ip} = $temp[6];
        $status{$ip} = "Incorrect OIU" unless $bad_omu{$ip} == 1;
    } elsif (/^Incorrect OMU/) {
        $ip = $temp[5];
        chop $temp[6];
        $bad_omu{$ip} = 1;
        $omu_name{$ip} = $temp[6];
        $status{$ip} = "Incorrect OMU";
    } elsif (/OK/) {
        $ip = $temp[0];
        chop $temp[1];
        $omu_name{$ip} = $temp[1];
```

```
            $status{$ip} = "All versions OK";
        }
    }

    @iplist = sort { "$a" <=> "$b" } (keys %omu_name);
    foreach (@iplist) {
        print DB "$_,$omu_name{$_},$status{$_}\n";
    }

    print "checkverdb.perl complete.\n";
    exit;
```